An Efficient LLL Gram Using Buffered Transformations

Werner Backes and Susanne Wetzel
{wbackes,swetzel}@cs.stevens.edu

STEVENS
Institute of Technology

Overview

- Introduction
- Preliminaries
- LLL Algorithm
  - Schnorr-Euchner
  - Gram Variant
  - Optimizations
  - Related Work
  - Results
- Conclusion/Future Work

Lattice

Definition

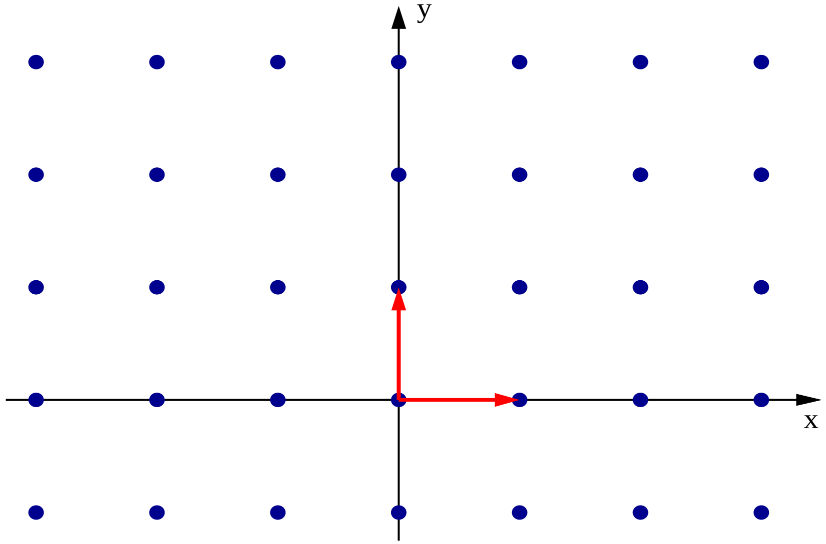
Let $n, k \in \mathbb{Z}$ with $k \leq n$. A **lattice** $L \subset \mathbb{R}^{\mathrm{n}}$ is a discrete, additive subgroup of $\mathbb{R}^{\mathrm{n}}$, such that

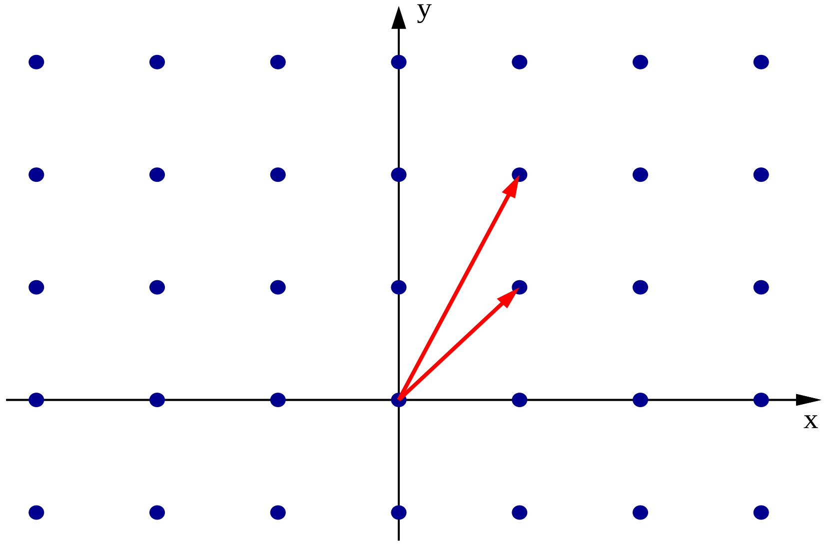
$$L = \{\sum_{i=1}^{k} x_i \underline{b}_i \mid x_i \in \mathbb{Z}, i = 1, \ldots, k\},$$

where $\underline{b}_1, \underline{b}_2, \ldots, \underline{b}_k \in \mathbb{R}^{\mathrm{n}}$ are linearly independent vectors.

$B = (\underline{b}_1, \ldots, \underline{b}_k) \in \mathbb{R}^{\mathrm{n \times k}}$ is a **basis** of the k-dimensional lattice L.

Example





$$L_1 = \left\{ \begin{pmatrix} x \\ y \end{pmatrix} \mid \begin{pmatrix} x \\ y \end{pmatrix} = c \begin{pmatrix} 1 \\ 0 \end{pmatrix} + d \begin{pmatrix} 0 \\ 1 \end{pmatrix} ; c, d \in \mathbb{Z} \right\}$$

$$L_2 = \left\{ \begin{pmatrix} x \\ y \end{pmatrix} \mid \begin{pmatrix} x \\ y \end{pmatrix} = c \begin{pmatrix} 1 \\ 1 \end{pmatrix} + d \begin{pmatrix} 1 \\ 2 \end{pmatrix} ; c, d \in \mathbb{Z} \right\}$$

Preliminaries

Lattice basis reduction: Find a basis $B' = (\underline{b}'_1, \ldots, \underline{b}'_k)$ for lattice $L(B)$ with $BU = B'$ (U unimodular) such that the basis vectors are as short and as orthogonal to each other as possible.

Gram-Schmidt orthogonalization:

$$\underline{b}_1^* = \underline{b}_1 \qquad \underline{b}_i^* = \underline{b}_i - \sum_{j=1}^{i-1} \mu_{ij} \underline{b}_j^* \qquad \text{for } 2 \leq i \leq k$$

$$\mu_{ij} = \frac{\langle \underline{b}_i, \underline{b}_j^* \rangle}{\|\underline{b}_j^*\|^2} \qquad \text{for } 1 \leq j < i \leq k$$

Definition (LLL-Reduction)

A basis $B = (\underline{b}_1, \ldots, \underline{b}_k)$ of the lattice $L \subset \mathbb{R}^n$ is called LLL-*reduced* for reduction parameter $\frac{1}{4} < \delta < 1$ if the following holds:

$$|\mu_{ij}| \leq \frac{1}{2} \qquad \text{for } 1 \leq j < i \leq k$$

$$\|\underline{b}_i^* + \mu_{ii-1} \underline{b}_{i-1}^*\|^2 \geq \delta \|\underline{b}_{i-1}^*\|^2 \qquad \text{for } 1 < i \leq k$$

LLL Algorithm – Schnorr-Euchner

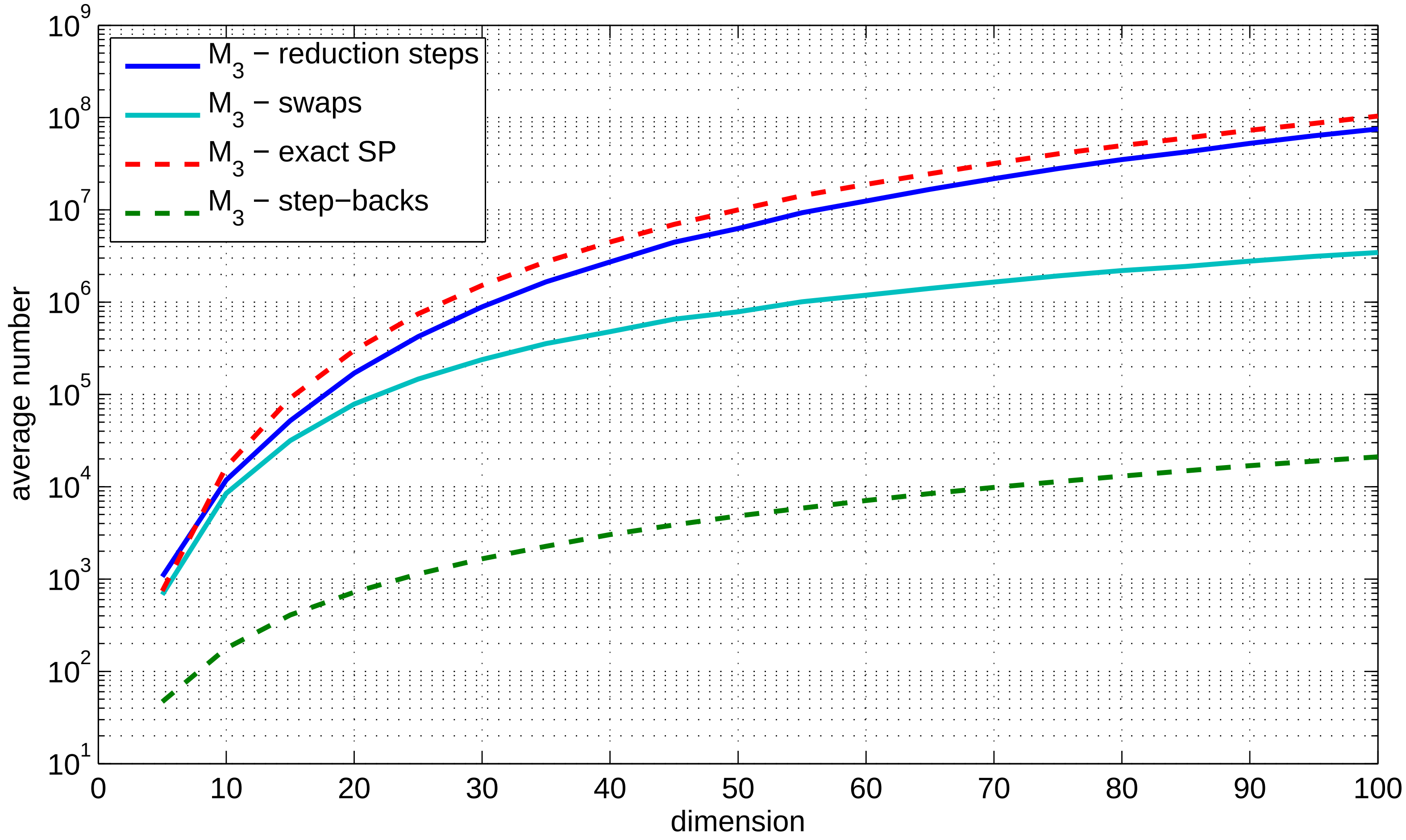
INPUT: Lattice basis $B = (\underline{b}_1, \ldots, \underline{b}_k) \in \mathbb{Z}^{n \times k}$, δ

OUTPUT: LLL-reduced lattice basis

$\mathrm{APPROX\_BASIS}(B', B)$
$B_1 = \|\underline{b}'_1\|^2$, $i = 2$
$F_c = false$, $F_r = false$
while $(i \le k)$ **do**
  $B_i = \|\underline{b}'_i\|^2$
  for $(2 \le j \le i)$ **do**
    if $(|\langle \underline{b}'_i, \underline{b}'_j \rangle| < 2^{\frac{r}{2}} \|\underline{b}'_i\| \|\underline{b}'_j\|)$ **then**
      $s = \mathrm{APPROX\_VALUE}(\langle \underline{b}_i, \underline{b}_j \rangle)$
    else
      $s = \langle \underline{b}'_i, \underline{b}'_j \rangle$
    $\mu_{ij} = (s - \sum_{m=1}^{j-1} \mu_{jm} \mu_{im} B_m) / B_j$
    $B_i = B_i - \mu_{ij}^2 B_j$
  $\mu_{ii} = 1$
  for $(i > j \ge 1)$ **do**
    if $(|\mu_{ij}| > \frac{1}{2})$ **then**
      $F_r = true$
      if $(|\lceil \mu_{ij} \rfloor| > 2^{\frac{r}{2}})$ **then**
        $F_c = true$
      $\underline{b}_i = \underline{b}_i - \lceil \mu_{ij} \rfloor \underline{b}_j$
      for $(1 \le m \le j)$ **do**
        $\mu_{im} = \mu_{im} - \lceil \mu_{ij} \rfloor \mu_{jm}$
  if $(F_r = true)$ **then**
    $\mathrm{APPROX\_VECTOR}(\underline{b}'_i, \underline{b}_i)$
    $F_r = false$
  if $(F_c = true)$ **then**
    $i = \max\{i-1, 2\}$
    $F_c = false$
  else
    if $(B_i < (\delta - \mu_{i\,i-1}^2) B_{i-1})$ **then**
      $\mathrm{SWAP}(\underline{b}_{i-1}, \underline{b}_i)$
      if $(i = 2)$ **then**
        $B_1 = \|\underline{b}'_1\|^2$
      $i = \max\{i-1, 2\}$
    else
      $i = i + 1$

exact scalar products, step-backs, reduction steps, swaps

LLL Algorithm – Correction Steps



- Correction steps (exact scalar products and step-backs) affect running time
- Exact scalar products have bigger impact.

LLL Algorithm – Gram Matrix

Definition

For a lattice L with basis $B = (\underline{b}_1, \ldots, \underline{b}_k) \in \mathbb{R}^{\mathrm{n} \times \mathrm{k}}$, the corresponding Gram matrix G is defined as $G = B^T B$.

- Gram matrix is the matrix of scalar products, therefore symmetric

Advantage:

- Avoid expensive exact scalar products.

Problem:

- Gram matrix not of interest: update of Gram matrix and lattice basis necessary.

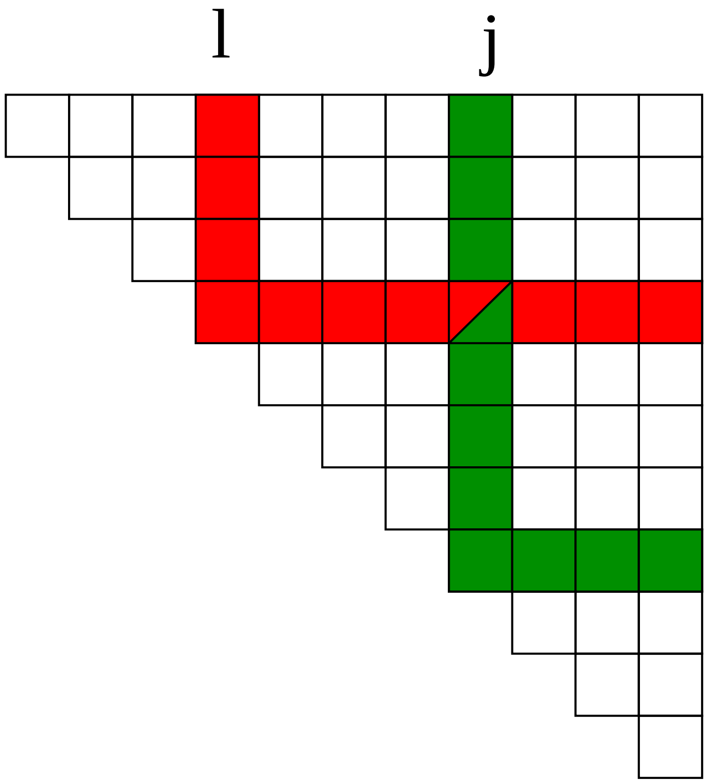
LLL Algorithm – Gram Variant

INPUT: Lattice basis $B = (\underline{b}_1, \ldots, \underline{b}_k) \in \mathbb{Z}^{n \times k}$, δ

OUTPUT: LLL-reduced lattice basis

```
COMPUTE_GRAM(A, B)
APPROX_BASIS_GRAM(A', A)
R_11 = A'_11, i = 2
F_c = false, F_r = false
while (i ≤ k) do
  S_1 = R_ii
  for (2 ≤ j ≤ i) do
    R_ij = A'_ji − Σ_{m=1}^{j−1} R_im μ_im
    μ_ij = R_ij / R_jj
    R_ii = R_ii − R_ij μ_ij
    S_{j+1} = R_ii
  μ_ii = 1
  for (i > j ≥ 1) do
    if (|μ_{i,j}| > 1/2) then
      F_r = true
      if (|μ_ij| > 2^{r/2}) then
        F_c = true
      b_i = b_i − ⌈μ_ij⌋ b_j
      REDUCE_GRAM(A, i, ⌈μ_ij⌋, j)
      for (1 ≤ m ≤ j) do
        μ_im = μ_im − ⌈μ_ij⌋ μ_im
  if (F_r = true) then
    APPROX_VECTOR_GRAM(A', A, i)
    F_r = false
  if (F_c = true) then
    i = max(i − 1, 2)
    F_c = false
  else
    i' = i
    while ((i > 1) ∧ (δ · R_{(i−1)(i−1)} > S_{i−1})) do
      SWAP(b_i, b_{i−1})
      SWAP_GRAM(A, i − 1, i)
      i = i − 1
    if (i ≠ i') then
      if (i = 1) then
        R_11 = A'_11, i = 2
    else
      i = i + 1
```

LLL Algorithm – Gram Variant



Algorithm: $\mathrm{REDUCE\_GRAM}(A, l, \lceil \mu_{ij} \rfloor, j)$

INPUT: Gram matrix A, indices l, j, $\lceil \mu_{ij} \rfloor$
OUTPUT: Gram matrix A

(1) $T = A_{l,l} - 2 \cdot \lceil \mu_{ij} \rfloor \cdot A_{j,l} - \lceil \mu_{ij} \rfloor^2 \cdot A_{j,j}$
(2) **for** $(1 \leq m < j)$ **do**
(3) $\quad A_{m,l} = A_{m,l} - \lceil \mu_{ij} \rfloor \cdot A_{m,j}$
(4) **for** $(j \leq m < l)$ **do**
(5) $\quad A_{m,l} = A_{m,l} - \lceil \mu_{ij} \rfloor \cdot A_{j,m}$
(6) **for** $(l + 1 \leq m < k)$ **do**
(7) $\quad A_{l,m} = A_{l,m} - \lceil \mu_{ij} \rfloor \cdot A_{j,m}$
(8) $A_{l,l} = T$

LLL Algorithm – Optimizations (1)

1. Buffer transformation to lattice basis using machine integers

Algorithm: $\text{BUFFERED\_TRANSFORM}(B, i, \lceil\mu_{ij}\rfloor, j)$

INPUT: Lattice Basis $B = (\underline{b}_1, \ldots, \underline{b}_k) \in \mathbb{Z}^{n\times k}$, indices i, j, $\lceil\mu_{ij}\rfloor$
OUTPUT: Lattice Basis B

if $((Tmax_i + |\lceil\mu_{ij}\rfloor| Tmax_j) \geq 2^{m-1})$ **then**
    for $(pos_{min} \leq x \leq pos_{max})$ **do**
        for $(1 \leq z \leq n)$ **do**
            $B'_{xz} = 0$
        for $(pos_{min} \leq y \leq pos_{max})$ **do**
            for $(1 \leq z \leq n)$ **do**
                $B'_{xz} = B'_{xz} + T_{xy} \cdot B_{yz}$
    $B \leftrightarrow B'$
    $T = I_n$
    $Tmax = (1, \ldots, 1)^T$
    $pos_{max} = i$
    $pos_{min} = j$
    if $(|\lceil\mu_{ij}\rfloor| > 2^{m-1} - 1)$ **then**
        $\underline{b}_i = \underline{b}_i - \lceil\mu_{ij}\rfloor \cdot \underline{b}_j$
        return
$\underline{t}_i = \underline{t}_i - \lceil\mu_{ij}\rfloor \cdot \underline{t}_j$
$Tmax_i = Tmax_i + |\lceil\mu_{ij}\rfloor| \cdot Tmax_j$
if $(pos_{max} < i)$ **then**
    $pos_{max} = i$
if $(pos_{min} > j)$ **then**
    $pos_{min} = j$

LLL Algorithm – Optimizations (2)

2. Avoid expensive operations (Multiplications)
3. Vectorization (Multimedia Streaming Extensions)

$\ldots$

if $(T_{xy} \neq 0)$ **then**
  if $(T_{xy} = 1)$ **then**
    for $(1 \leq z \leq n)$ **do**
      $B'_{xz} = B'_{xz} + B_{yz}$
  else
    if $(T_{xy} = -1)$ **then**
      for $(1 \leq z \leq n)$ **do**
        $B'_{xz} = B'_{xz} - B_{yz}$
    else
      for $(1 \leq z \leq n)$ **do**
        $B'_{xz} = B'_{xz} + T_{xy} \cdot B_{yz}$

$\ldots$

$\ldots$

for $(1 \leq l \leq n; l+ = 4)$ **do**
  $T_{i,l} = T_{i,l} - \lceil \mu_{ij} \rfloor \cdot T_{j,l}$
  $T_{i,l+1} = T_{i,l+1} - \lceil \mu_{ij} \rfloor \cdot T_{j,l+1}$
  $T_{i,l+2} = T_{i,l+2} - \lceil \mu_{ij} \rfloor \cdot T_{j,l+2}$
  $T_{i,l+3} = T_{i,l+3} - \lceil \mu_{ij} \rfloor \cdot T_{j,l+3}$

$\ldots$

LLL Algorithm – Related Work

NTL (by Victor Shoup)

- based on Schnorr-Euchner, with lots of improvements.
- modified heuristic to reduce number of exact scalar products

fpLLL (by Damien Stehlé and Phong Nguyen)

- uses Gram matrix
- new (δ, η) reduction condition with $\delta \in (\frac{1}{2}, 1)$ and $\eta > \frac{1}{2}$
- provable, but weaker than original LLL ($\eta = 0.5$)

Segment LLL-Reduction (by Schnorr Group)

- weaker reduction condition
- Householder orthogonalization

LLL Algorithm – Results (1)

Lower triangular matrices U_j, upper triangular matrices V_j with $1 \leq j \leq 2$, permutation matrices P_i for $1 \leq i \leq 4$, create three types of unimodular lattice bases:

- M_1: $B = (U_1 \cdot V_1)$
- M_2: $B = (U_1 P_1 \cdot V_1 P_2)$
- M_3: $B = (U_1 P_1 \cdot V_1 P_2) \cdot (V_2 P_3 \cdot U_2 P_4)$

Test Parameter:

- 1000 lattice bases for each type and dimension $5, 10, 15, \ldots, 100$
- maximal bit length of lattice basis entries is 500

LLL Algorithm – Results (2)

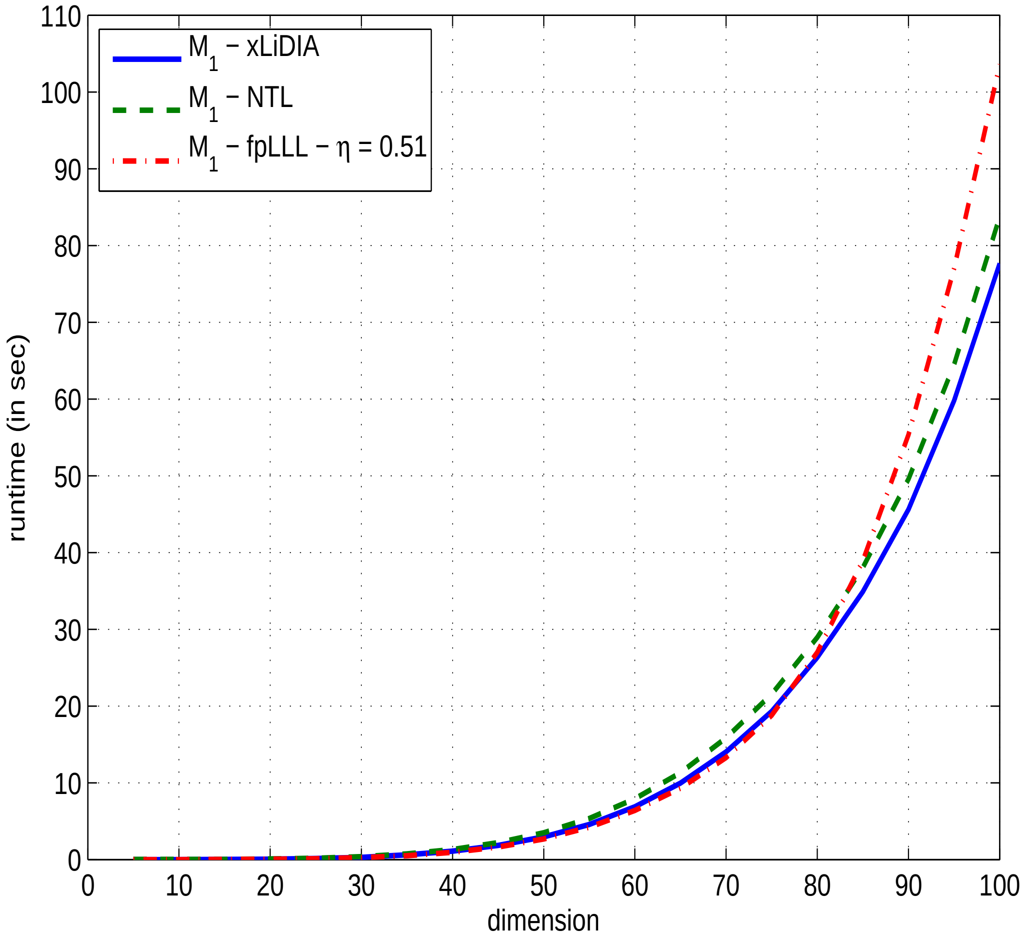
Test Setup:

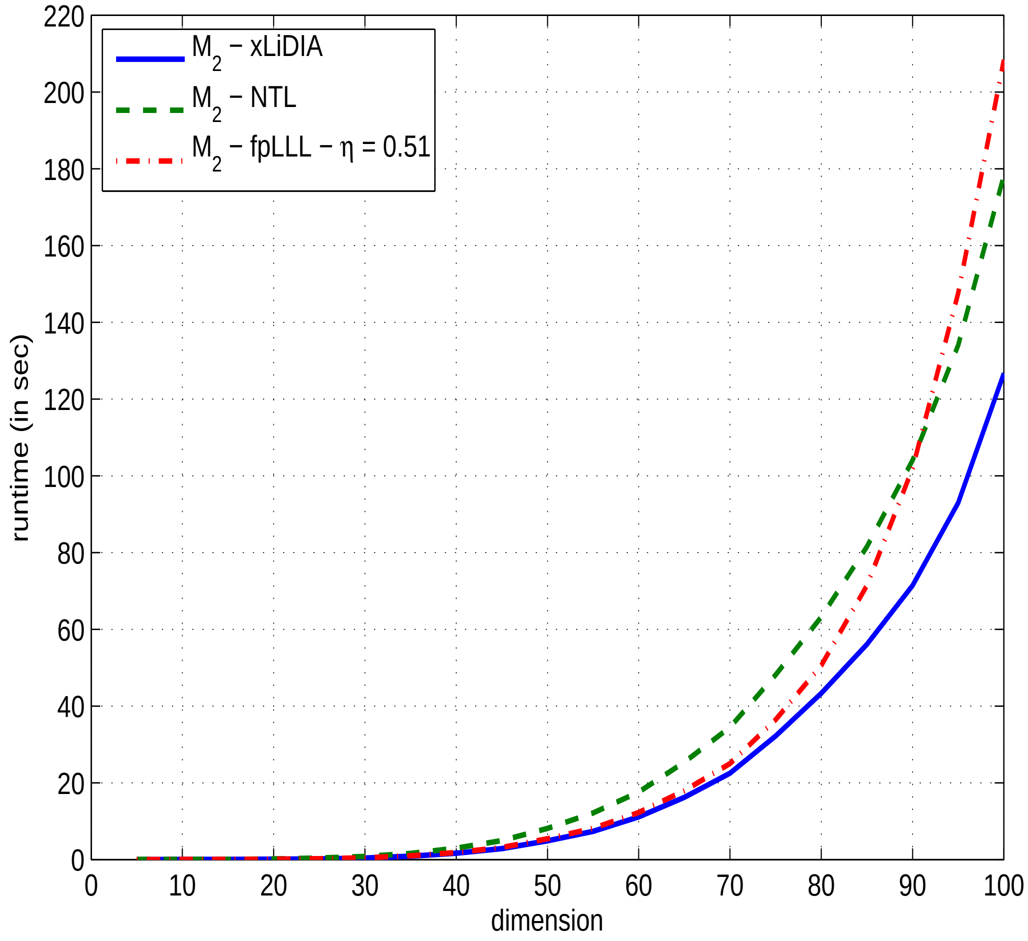
- LLL_FP from NTL 5.4 (by Victor Shoup)
- proved variant for fpLLL 1.3 with $\eta = 0.51$ (by Damien Stehlé and Phong Nguyen)
- xLiDIA implementation (modified, stripped down version of LiDIA)
- GCC 4.1, GMP 4.2.1 with additional AMD64 patch (by Pierrick Gaudry)

Test System:

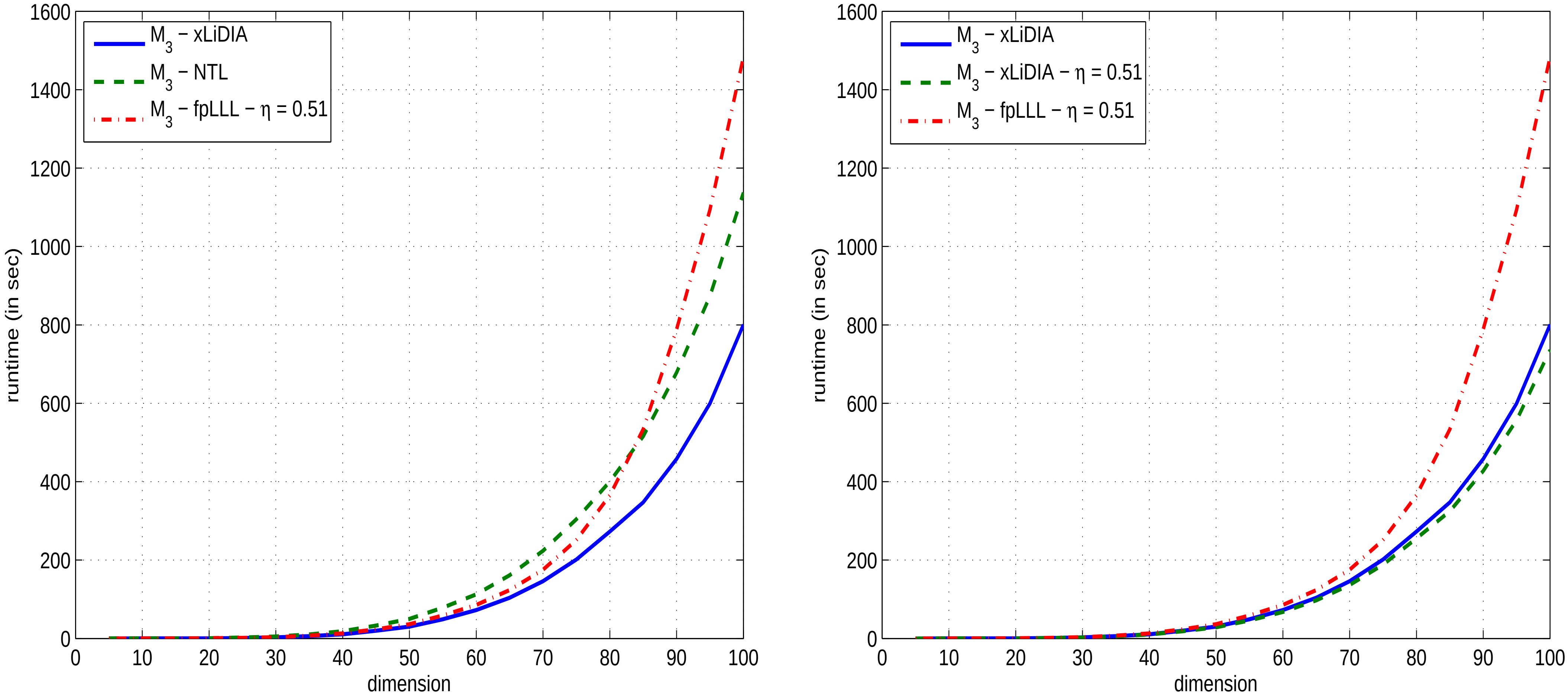
- Test System: Sun X4100 Server, AMD Opteron 2.2 GHz and 4GB RAM, Solaris 10

LLL Algorithm – Results (3)

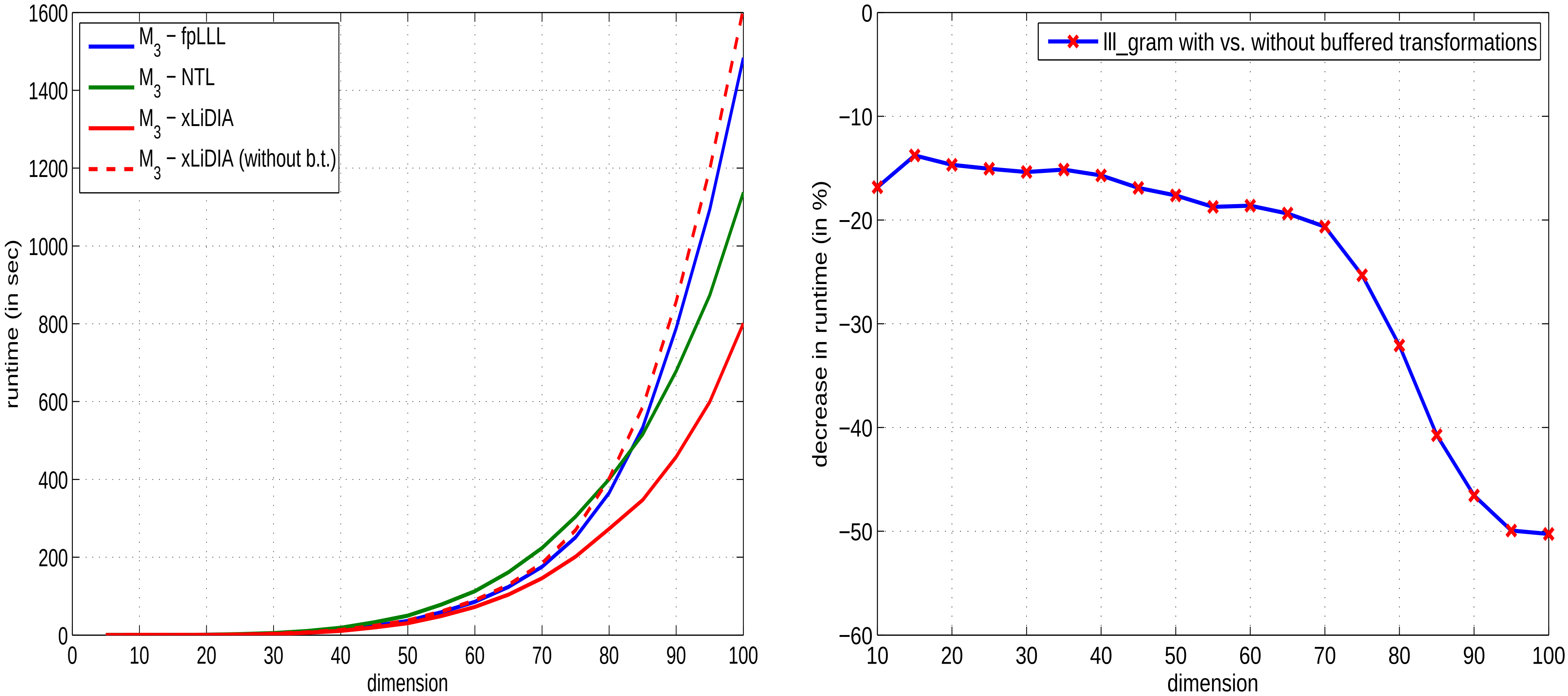




LLL Algorithm – Results (4)

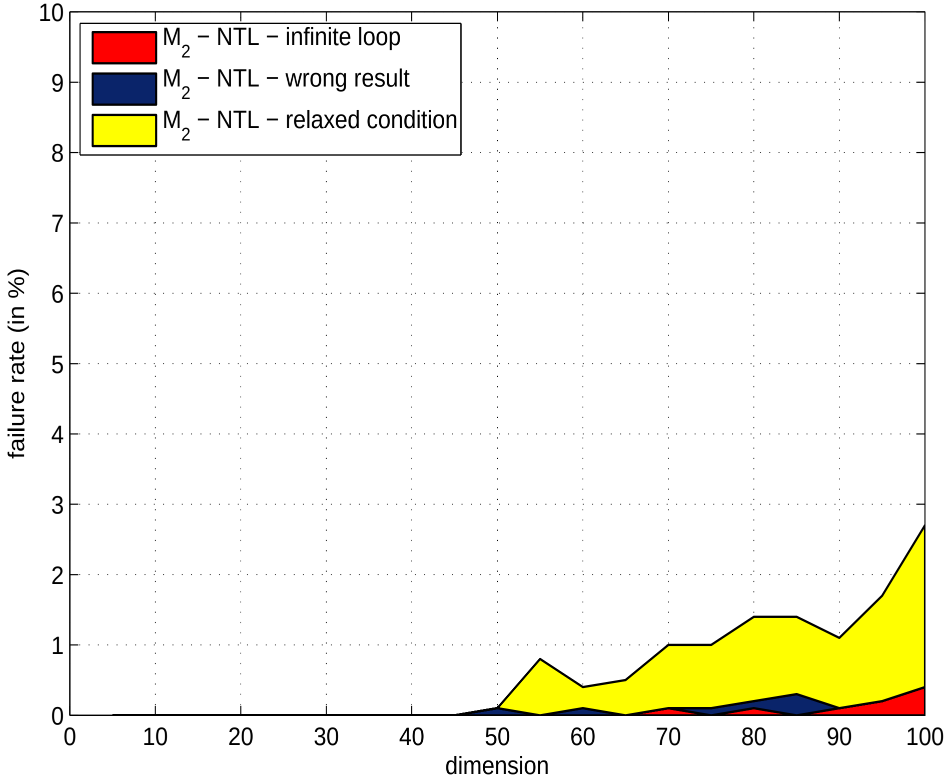


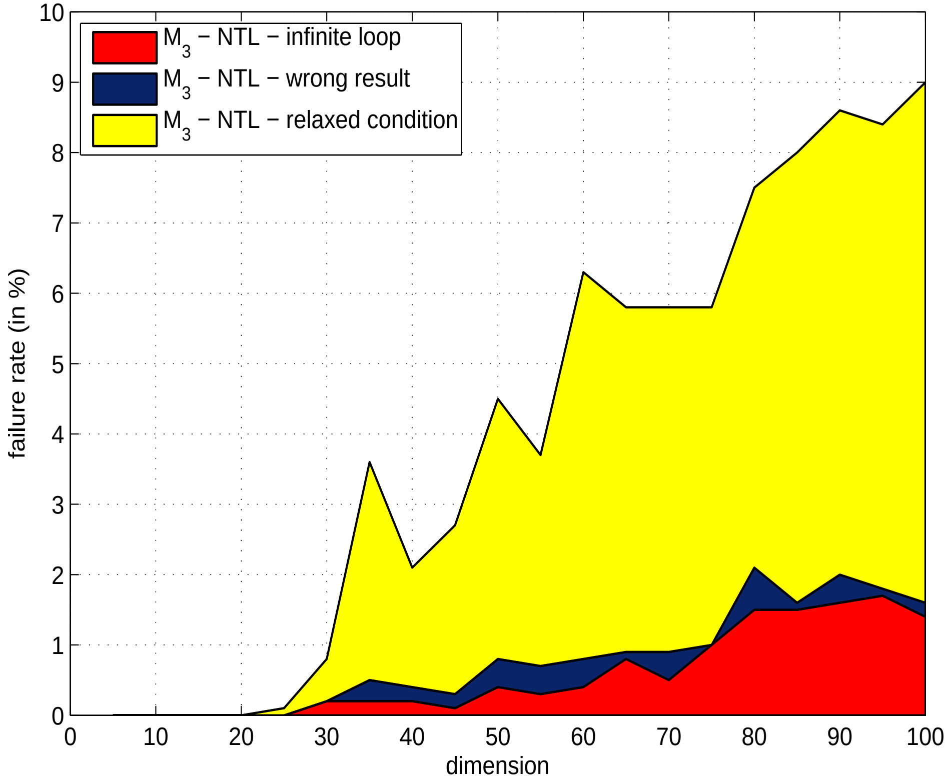
LLL Algorithm – Results (5)



- LLL Gram with and without buffered transformations.

LLL Algorithm – Results (6)





- `fast` and `heuristic` variant of fpLLL fails for small bases (dimension 10 with entries of maximum bit length of 100 bits)

Conclusion/Future Work

Conclusion

- Our implementation outperforms NTL and fpLLL
- Buffered transformation offer massive runtime improvement
- Keep original LLL condition (compared to fpLLL)
- Improved stability (compared to NTL)

Future Work

- Parallel version (multi-core CPUs)
- Extend use of machine-type doubles

References

[1] W. Backes and S. Wetzel. Heuristics on Lattice Basis Reduction in Practice. *ACM Journal on Experimental Algorithms*, 7, 2002.

[2] A. Lenstra, H. Lenstra, and L. Lovász. Factoring Polynomials with Rational Coefficients. *Math. Ann.*, 261:515–534, 1982.

[3] P. Nguyen and D. Stehlé. Floating-Point LLL Revisited. In *Proceedings of Eurocrypt 2005*, volume 3494 of *LNCS*, pages 215–233. Springer, 2005.

[4] C. Schnorr and M. Euchner. Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems. In *Proceedings of Fundamentals of Computation Theory '91*, volume 529 of *LNCS*, pages 68–85. Springer, 1991.

[5] D. Stehlé. The New LLL Routine in the Magma Computational Algebra System. *Magma 2006 Conference*. `http://magma.maths.usyd.edu.au/Magma2006/`.

[6] S. Wetzel. *Lattice Basis Reduction Algorithms and their Applications*. PhD thesis, Universität des Saarlandes, 1998.

[7] fpLLL - Homepage (Damien Stehlé), July 2007. `http://www.loria.fr/~stehle/`.

[8] AMD64 patch for GMP 4.2, July 2007. `http://www.loria.fr/~gaudry/mpn_AMD64/index.html`.

[9] NTL - Homepage, July 2007. `http://www.shoup.net/ntl/`.